1989

Some applications of fast Fourier transforms

Mustafa Kizilkaya
Lehigh University

SOME APPLICATIONS OF FAST FOURIER TRANSFORMS

by

MUSTAFA KIZILKAYA

A Thesis

Presented to the Graduate Committee

of Lehigh University

in Candidacy for the Degree of

Master of Science

in

Applied Mathematics

(In the Department of

Mechanical Engineering and Mechanics)

Lehigh University

1989

This thesis is accepted and approved in partial fulfillment of the requirements for the degree of Master of Science.

May 8, 1989

(date)

Jacob Y. Korbahn

Professor in Charge

F. Erdogan

Chairman of the Department

ACKNOWLEDGEMENT

I would like to express my sincerest appreciation to my advisor Professor Jacob Y. Kazakia, for his guidance, interest and extreme patience in overseeing the completion of this work. I thank the Turkish Government for paying my tuition, salary and all expenses during my studies at Lehigh. I would also like to thank my Mother, sister and brothers for their love.

TABLE OF CONTENTS

List of Tables

List of Figures

Abstract

The use of Fast Fourier Transforms in solving partial differential equations is studied. The basic theory of FFT is summarized and efficient subroutines for the fast transformation of one or two dimensional data arrays are described .

The methods of solution for Poisson's equation with periodic or Dirichlet boundary conditions are studied in detail.

A specific example on the Laplace's equation over a square region with homegeneous boundary conditions on three sides and a constant inhomogeneous condition on the fourth is worked out. The results obtained by FFT and the results obtained by Simultaneous Over-Relaxation (SOR) of the finite difference equations are compared with the analytic solution. It is cancluded that the FFT method is at least as accurate as SOR but significantly faster.

1. Introduction

In recent years significant progres has been made in incorporating Fast Fourier Transform to the solution of partial differential equations.

Efficient software and associated special hardware have increased dramatically the speed of discrete Fourier transformation making the methods associated with it (Spectral methods in general) the method of choise for several practical problems.

The Fast Fourier Transform (FFT) algorithm is a method for computing the finite Fourier transform of a series of N (complex) data points in approximately $N\log_2 N$ operations. The algorithm has a fasinating history. When it was described by Cooley and Tukey in 1965 it was regarded as new by many knowledgeable people who believed Fourier analysis to be a process requiring something proportional to N^2 operations with a proportionality factor which could be reduced by using the symmetries of the trigonometric functions. Computer programs using the N^2 operation methods were, in fact, using up hundreds of hours of machine time.

It was shown how one could use the *periodicity* of the sine-cosine functions to obtain a $2N$-point Fourier analysis from two N-point analyses with only slightly more then N operations. Going the other way, if the series to be transformed is of length N and N is a power of 2, the series can be split into $\log_2 N$ subseries and this doubling algorithm can be applied to compute finite Fourier transform in $\log_2 N$ doublings. The number of computations in the resulting succesive doubling algorithm is therefore proportional to $N\log_2 N$ rather then N^2.

Fast Fourier transforms are used extensively in the area of digital processing of signals, where they simplify considerably the analysis of signals may they be voice, TV,

speach, music or radar. These fast techniques are very helpful in such tasks as identifying the speaker or sonar emitter and in reducing the data required to transmit a TV picture.

In this thesis we focus on the use of FFT in the solution two-dimensional boundary value problems. In section 2 we develop the basic concepts of Discrete Fourier Transform, and we describe the Cooly-Tukey algorithm, which makes the discrete transform fast. In section 3, we discuss the use of FFT to the solution of Poisson's equation. We consider periodic boundary condition in two and three dimensions. We also consider the problem with Dirichlet boundary conditions. A method utilizing a two dimensional Fast Sine Transform is explained and the routines associated with that are described. A specific example is worked out in section 4. We solve $\nabla^2 u=0$ over a square region with homegeneous conditions on the three sides and $u=1$ on the fourth side. We solve the problem using Simultaneous Over-Relaxation and Fast Sine Transform. The solutions are compared with the analytical solution for speed and accuracy. The results of the comparision are given in section 5. It is quite evident that the FFT is at least as acurate as SOR but significantly faster.

2. DISCRETE FOURIER TRANSFORMS (DFT)

The fast Fourier transform (FFT) method is a computational algorithm which, greatly increases the speed with which Fourier transforms can be computed on digital devices. As a consequence, digital application of Fourier methods has been widely utilized and can be expected to be used even more as the FFT method is incorporated in efficient computer subroutines, and special hardware. Therefore, it seems useful to investigate further the essential properties of the discrete Fourier transform (DFT), its correspondence with integral transforms, and various algorithms for using the discrete Fourier transform in special circumstances.

The discrete complex Fourier series is a one-to-one mapping of any sequence $x(n)$, n=0,1,...,N-1, of N complex numbers onto another sequence defined by

$$X(j)=\sum_{n=0}^{N-1} x(n)\, W_N^{\,nj} \quad , \quad j=0,1,...,N-1, \tag{2.1}$$

where $i=\sqrt{-1}$ and where $W_N=\exp(2\pi i/N)$ is the principal Nth complex root of unity.

The formula for $x(n)$ in terms of $X(j)$,

$$x(n)=\frac{1}{N}\sum_{j=0}^{N-1} X(j)\, W_N^{\,-nj}, \quad n=0,1,...,N-1 \tag{2.2}$$

is known as the inverse discrete Fourier transform (IDFT). A direct calculation of (2.1) as an accumulated sum of products for each j would take N^2 operations. The fast Fourier transform method (FFT) is an algoritm for computing (2.1) or (2.2) in $N\log N$ operations. The FFT algoritm has been described and programmed, in most cases, as a calculation of the operation defined by (2.1) on complex numbers. However, in actual practice, there is a wide variety of special conditions which one actually wants. For example, the data may be real rather then truly complex, and it may be even or odd so that cosine or sine transforms may be used. These transforms can be done directly with the complex DFT.

2.1. Matrix Representation of DFT

In this section we represent the DFT by a matrix, and rearranging the matrix we shall end up with a matrix factorization leading to the FFT. The input to the DFT is the data sequence contained in the vector $\underset{\sim}{x}$ given by

$$\underset{\sim}{x} = [\, x(0)\ ,\ x(1)\ ,\ \ldots\ ,\ x(N\text{-}1)\ ]^{T} \tag{2.3}$$

where the supercript T denotes the transpose. The output of the DFT is the transform sequence contained in the vector $\underset{\sim}{X}$ given by

$$\underset{\sim}{X} = [\, X(0)\ ,\ X(1)\ ,\ \ldots\ ,\ X(N\text{-}1)\ ]^{T} \tag{2.4}$$

The outputs are computed using the DFT definition . For example , if N = 4 , The definition of DFT gives

$$\underset{\sim}{X} = \hat{\underset{\sim}{W}}\,\underset{\sim}{x} \tag{2.5}$$

where $\hat{\underset{\sim}{W}}$ is the DFT matrix

$$W = \begin{bmatrix} W^0 & W^0 & W^0 & W^0 \\ W^0 & W^1 & W^2 & W^3 \\ W^0 & W^2 & W^0 & W^2 \\ W^0 & W^3 & W^2 & W^1 \end{bmatrix} \tag{2.6}$$

where, of course, $W^0 = 1,\ W^1 = i,\ W^2 = -1,\ W^3 = -i.$

We can get another way of representing the matrix $\hat{\underset{\sim}{W}}$ by noting that

$$\hat{W}_{jk} = W^{E_{jk}}, \tag{2.7}$$

where the matrix $\underset{\sim}{E}$ is given by

$$\underset{\sim}{E} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 3 \\ 0 & 2 & 0 & 2 \\ 0 & 3 & 2 & 1 \end{bmatrix}. \tag{2.8}$$

More generally, the E matrix of dimension N is given by

$$E = \begin{bmatrix} 0 & 0 & 0 & . & 0 & 0 \\ 0 & 1 & 2 & . & N-2 & N-1 \\ 0 & 2 & 4 & . & N-4 & N-2 \\ . & . & . & . & . & . \\ 0 & N-2 & N-4 & . & 4 & 2 \\ 0 & N-1 & N-2 & . & 2 & 1 \end{bmatrix} \tag{2.9}$$

In conclusion, (2.5) is the vector-matrix equation for the DFT. The matrix of exponents is given by (2.9).

Another way of introducing the discrete Fourier transform is obtained via the polynomial

$$P(x) = \sum_{k=0}^{n-1} p_k \; x^k, \tag{2.10}$$

where p_k are the coefficents of the polynomial. The values of $P(x)$ for $x = W^0, W^1, \ldots, W^{N-1}$ form the DFT of the coefficients p_k. *i.e.*

$$\underset{\sim}{P} = \underset{\sim}{F}\,\underset{\sim}{p} \tag{2.11}$$

where

$$\underset{\sim}{P} = [P(W^0), P(W^1), \ldots, P(W^{N-1})]^{\top}, \tag{2.12}$$

and

$$\underset{\sim}{p} = [p_1, p_2, \ldots, p_{N-1}]^{\top}, \tag{2.13}$$

and $\underset{\sim}{F}$ is the *Vandermonde matrix*:

$$\underset{\sim}{F} = \begin{bmatrix} (W^0)^0 & (W^0)^1 & . & (W^0)^{N-1} \\ (W^1)^0 & (W^1)^1 & . & (W^1)^{N-1} \\ . & . & . & . \\ (W^{N-1})^0 & (W^{N-1})^1 & . & (W^{N-1})^{N-1} \end{bmatrix} \tag{2.14}$$

Since $\underset{\sim}{F}$ is a *Vandermonde matrix* with distinct elements is nonsingular and $\underset{\sim}{F}^{-1}$ exists.

In fact, writing (2.14) in the form

$$F_{ij} = W^{i \bullet j} \tag{2.15}$$

we can see that

$$F_{ij}^{-1} = \frac{1}{N} W^{-(i \bullet j)}. \tag{2.16}$$

The above statement can be verified by forming the product

$$C_{ij} = \sum_{k=0}^{N-1} F_{ik} F^{-1}{}_{kj}$$

$$= \sum_{k=0}^{N-1} \frac{1}{N} W^{i \bullet k} W^{-(k \bullet j)}$$

$$= \frac{1}{N} \sum_{k=0}^{N-1} \alpha^k \tag{2.17}$$

where $\alpha = 1$ for $i = j$ and $\alpha = W^{i-j} \neq 1$ otherwise.

It is obvious that $C_{ii} = 1$ for all i, the fact that $C_{ij} = 0$ for $i \neq j$ can be seen by summing the geometric series (2.17) and obtaining

$$C_{ij} = \frac{1}{N} \frac{\alpha^N - 1}{\alpha - 1} \tag{2.18}$$

and noting that $\alpha^N = 1$, for all i and j.

2.2. The Fast Fourier Transform (FFT)

In general we can use the notation $\underset{\sim}{F}_N$ to indicate the DFT matrix for N points. In view of the explanation presented in the previous section.

We can write

$$\boxed{(\underset{\sim}{F}_N)_{jk} = W_N^{jk} = e^{2\pi ijk/N}}. \tag{2.19}$$

Where $i=\sqrt{-1}$. For example

$$F_2=\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \qquad F_4=\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{bmatrix}. \quad (2.20)$$

In a certain way F_4 connects back to F_2 and the whole inspiration of the Fast Fourier Transform is to find that connection! Similarly F_2 is related to F_4, and F_{80} is related to F_{40}.

2.2.1. Cooley–Tukey algorithm :

Our interest is in powers like $N=2^{12}$. There will be $N^2 = 2^{24}$ entries in F_N, and an ordinary matrix-vector product $F_N x$ requires 2^{24} complex multiplications. In itself that is not terrible; it takes a few seconds on a big machine. But if it is repeated thousands of times, as it is in time series analysis and image processing and elsewhere, the cost of these products $F_N x$ becomes prohibitive. By contrast, the Fast Fourier Transform finds $F_N x$ with only $6\cdot 2^{12}$ multiplications–it is more then 600 times faster. It replaces N^2 multiplications by $\frac{1}{2}N\log_2 N$.

The fast transform starts with a completely trivial observation : If N equals $2M$ then using equation (2.19) we see that

$$W_N{}^2 = W_M \text{ if } M=\tfrac{1}{2}N. \quad (2.21)$$

This identity allows $y=F_N x$ (a vector with N components) to be recovered from two vectors y' and y'' with N/2 components. That is the key idea. The vector ($x_0, x_1, \ldots, x_{N-1}$) is split into two shorter pieces, by separating the even and odd compnents:

$$\underset{\sim}{x}' = (x_0 , x_2, \dots , x_{N-2}) \text{ and } \underset{\sim}{x}'' = (x_1 , x_3, \dots , x_{N-1}) . \qquad (2.22)$$

From these vectors we form

$$\underset{\sim}{y}' = \underset{\sim}{F}_M \underset{\sim}{x}' \text{ and } \underset{\sim}{y}'' = \underset{\sim}{F}_M \underset{\sim}{x}'' . \qquad (2.23)$$

Those multiplications involve the half-size matrix $\underset{\sim}{F}_M$, but the results $\underset{\sim}{y}'$ and $\underset{\sim}{y}''$ contain enough infomation to reconstruct $\underset{\sim}{y}$.

It can be proven that the first M and the last M componens of $\underset{\sim}{y} = \underset{\sim}{F}_N \underset{\sim}{x}$ are

$$y_j = y'_j + (W_N)^j y''_j , \qquad j = 0, \dots , M-1$$

$$y_{j+m} = y'_j - (W_N)^j y''_j , \qquad j=0, \dots , M-1 . \qquad (2.24)$$

Thus the three steps needed for FFT *are* : split $\underset{\sim}{x}$ into $\underset{\sim}{x}'$ and $\underset{\sim}{x}''$, form $\underset{\sim}{y}' = \underset{\sim}{F}_M \underset{\sim}{x}'$ and $\underset{\sim}{y}'' = \underset{\sim}{F}_M \underset{\sim}{x}''$, and compute $\underset{\sim}{y}$ from (2.24).

Proof: The verification of (2.24) begins by dividing each component of $\underset{\sim}{y} = \underset{\sim}{F}_N \underset{\sim}{x}$ into odd end even parts. The entries of $\underset{\sim}{F}_N$ are powers of W_N, and the separation of $\underset{\sim}{y}$ follows exactly the separtion of $\underset{\sim}{x}$:

$$y_j = \sum_{k=0}^{N-1} W_N^{kj} x_k = \sum_{k=0}^{M-1} W_N^{2kj} x_{2k} + \sum_{k=0}^{M-1} W_N^{(2k+1)j} x_{2k+1} , \quad j = 0, \dots , N-1 \qquad (2.25)$$

Each sum on the right has M terms. The first involves the even components x_{2k} and comes from $\underset{\sim}{x}'$. The secont contains the components x_{2k+1} of $\underset{\sim}{x}''$. Since $W_N^2 = W_M$, we have

$$y_j = \sum_{k=0}^{M-1} W_M^{kj} x_k' + (W_N)^j \sum_{k=0}^{M-1} W_M^{kj} x_k'' . \qquad (2.26)$$

using (2.23) we have

$$y_j = y_j' + (W_N)^j y_j'' , \qquad j=0, \dots , M-1, \qquad (2.27)$$

which is the first part of (2.24). For the second part we write

$$y_{j+M} = \sum_{k=0}^{N-1} W_N^{k(j+M)} x_k$$

$$= \sum_{k=0}^{M-1} W_N^{2k(j+M)} x_{2k} + \sum_{k=0}^{M-1} W_N^{(2k+1)(j+M)} x_{2k+1}$$

$$= \sum_{k=0}^{M-1} W_M^{k(j+M)} x_k' + W_N^{j+M} \sum_{k=0}^{M-1} W_M^{k(j+M)} x_k''$$

$$= \sum_{k=0}^{M-1} W_M^{kj} W_M^{kM} x_k' + W_N^{j} W_N^{M} \sum_{k=0}^{M-1} W_M^{kj} W_M^{kM} x_k''. \qquad (2.28)$$

Since inside the sums,

$$W_M^{kM} = e^{(2\pi i/M)^{kM}} = (e^{2\pi i})^k = 1^k = 1, \qquad (2.29)$$

and outside,

$$W_N^{M} = (e^{2\pi i/N})^M = (e^{2\pi i/2M})^M = e^{\pi i} = -1, \qquad (2.30)$$

we have

$$y_{j+M} = \sum_{k=0}^{M-1} W_M^{kj} x_k' - (W_N)^j \sum_{k=0}^{M-1} W_M^{kj} x_k'', \qquad j = 0, \ldots, M-1, \qquad (2.31)$$

or

$$y_{j+M} = y_j' - (W_N)^j y_j''. \qquad (2.32)$$

Repeating the same idea, a similar rule connects M to $M/2$, $M/2$ to $M/4$ and so on *e.g*;

$$F_N \rightarrow F_{N/2} \rightarrow F_{N/4} \rightarrow F_{N/8} \rightarrow \ldots \rightarrow F_2. \qquad (2.33)$$

Let L_l represent the number of multiplication required for the FFT of size $N=2^l$. It can be shown by inspection of equations (2.24) that

$$\begin{aligned} (N=2) \qquad & L_1 = 2^2 \\ (N=4) \qquad & L_2 = 2\left\{2^2 + 2^1\right\} \\ (N=8) \qquad & L_3 = 2\left\{2\left\{2^2 + 2^1\right\} + 2^2\right\} \end{aligned} \qquad (2.34)$$

$$(N=16) \quad L_4=2\left\{2\left\{2\left\{2^2+2^1\right\}+2^2\right\}+2^3\right\}$$

. . . . . .

From (2.34) we can see that

$$L_l=2^{l+1}+(l-1)2^l=2^l(l+1). \tag{2.35}$$

We summarize the Cooly-Tukey algorithm by describing in detail the following example.

EXAMPLE

We need the DFT of the vector

$$\underset{\sim}{x}=[2,4,6,8]^T. \tag{2.36}$$

We could have calculated the transform $\underset{\sim}{y}$ from

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{bmatrix} \begin{bmatrix} 2 \\ 4 \\ 6 \\ 8 \end{bmatrix} = \begin{bmatrix} 20 \\ \text{-4-4}i \\ \text{-4} \\ \text{-4+4}i \end{bmatrix}. \tag{2.37}$$

This would have required 16 multiplications. According to Cooley-Tukey algorithm we will proceed as follows.

We split this vector $\underset{\sim}{x}$ into

$$\underset{\sim}{x}'=[\ 2\ ,\ 6\ ]^T, \quad \underset{\sim}{x}''=[\ 4\ ,\ 8\ ]^T \tag{2.38}$$

and we calculate $\underset{\sim}{y}'$, $\underset{\sim}{y}''$ by

$$\underset{\sim}{y}'=\begin{bmatrix} 1 & 1 \\ 1 & \text{-1} \end{bmatrix}\begin{bmatrix} 2 \\ 6 \end{bmatrix}=\begin{bmatrix} 8 \\ \text{-4} \end{bmatrix}, \quad \underset{\sim}{y}''=\begin{bmatrix} 1 & 1 \\ 1 & \text{-1} \end{bmatrix}\begin{bmatrix} 4 \\ 8 \end{bmatrix}=\begin{bmatrix} 12 \\ \text{-4} \end{bmatrix} \tag{2.39}$$

This step requires $2x2^2=8$ multiplication. We than use the formula (2.24) to obtain

$$y_j = y'_j + (W_4)^j y''_j ,$$
$$y_{j+2} = y'_j - (W_4)^j y''_j \qquad (2.40)$$

with $j=0,1$.

Considering that $(W_4)^0=1, (W_4)^1=i$ we have

$$y_0=8 + 12 \ , \ y_1= -4-4i \ , \ y_2= 8 - 12 \ , \ y_3= -4 + 4i.$$

The last step requires 4 multiplication. Consequently the total number of multiplication is $8+4=12$.

3. Application of FFT in Differential Equations

3.1. Periodic Difference Equations

The Fast Fourier Transform can solve certain finite difference equations with exceptional speed. To operate perfectly it needs a constant-coefficient problem $\underset{\sim}{K}\underset{\sim}{u}=\underset{\sim}{f}$ on a regular mesh.

Let us consider the central-difference equation

$$-u_{k+1} + du_k - u_{k-1} = f_k, \quad k=0,1,\ldots,N-1 \tag{3.1}$$

over a one dimensional mesh with the solution u having periodicity of N. For $N=4$ we have

$$\begin{aligned} -u_1+du_0-u_{-1}&=f_0 \\ -u_2+du_1-u_0&=f_1 \\ -u_3+du_2-u_1&=f_2 \\ -u_4+du_3-u_2&=f_3 \end{aligned} \tag{3.2}$$

and considering that $u_{-1}=u_3$ and $u_4=u_0$ we write

$$\begin{bmatrix} d & -1 & 0 & -1 \\ -1 & d & -1 & 0 \\ 0 & -1 & d & -1 \\ -1 & 0 & -1 & d \end{bmatrix} \begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \end{bmatrix} = \begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ f_3 \end{bmatrix} \tag{3.3}$$

In general we want to solve the equations

$$\underset{\sim}{T}_N\underset{\sim}{u}=\underset{\sim}{f}, \qquad \underset{\sim}{u}=[\,u_0,u_1,\ldots,u_{N-1}]^T \tag{3.4}$$

This is a one-dimensional diference equation and it does not actually need the FFT. However in two dimensions fast transforms will make all the difference. The main

point is :

$\underset{\sim}{T}_N$ is diagonalized by the Fourier matrix $\underset{\sim}{F}_N$, *so that* $\underset{\sim}{F}_N{}^{-1}\underset{\sim}{T}_N\underset{\sim}{F}_N=\underset{\sim}{\Lambda}_N$.

where

$$\underset{\sim}{T}_N=\begin{bmatrix} d & -1 & 0 & . & -1 \\ -1 & d & -1 & . & 0 \\ 0 & -1 & d & . & 0 \\ . & . & . & . & -1 \\ -1 & 0 & 0 & -1 & d \end{bmatrix}, \qquad \underset{\sim}{F}_N=\left(W_N{}^{jk} \right), \tag{3.5}$$

and $\underset{\sim}{\Lambda}_N$ is a diagonal matrix with

$$[\underset{\sim}{\Lambda}_N]_{jj}=\lambda_j=\left(d-W_N{}^j-\frac{1}{W_N{}^j}\right). \tag{3.6}$$

This means that the columns of the matrix $\underset{\sim}{F}_N$ are the eigenvectors of the matrix $\underset{\sim}{T}_N$.

To the M^{th} eigenvalue λ_M given by (3.6), we have the corresponding eigenvector

$$(\underset{\sim}{k}_M)_j= W_N{}^{Mj}, \quad i.e.$$

$$\underset{\sim}{k}_M=\begin{bmatrix} 1 \\ W^M \\ W^{2M} \\ . \\ W^{NM} \end{bmatrix}. \tag{3.7}$$

As an example in the case $N=4$, with $W_4=i$, the eigenvalues will be $\lambda_0=d-1-1$, $\lambda_1=d-i-i^{-1}$, $\lambda_2=d-i^2-i^{-2}$, and $\lambda_3=d-i^3-i^{-3}$. The diagonal form becomes

$$F_4^{-1} T_4 F_4 = \Lambda_4 = \begin{bmatrix} d\text{-}2 & 0 & 0 & 0 \\ 0 & d & 0 & 0 \\ 0 & 0 & d+2 & 0 \\ 0 & 0 & 0 & d \end{bmatrix}. \tag{3.8}$$

Now we can solve the equation (3.4) using FFT. They allow a quick multiplication by F or F^{-1}, and the solution $u = T^{-1} f = F \Lambda^{-1} F^{-1} f$ is found in three steps:

(1) transform f *to* $F^{-1} f$

(2) divide by the eigenvalues to obtain $\Lambda^{-1} F^{-1} f$

(3) transform back to reach $F \Lambda^{-1} F^{-1} f$.

Note that step 2 fails when one of the eigenvalues is zero. This may happen in the important case $d=2$ which gives the finite difference approximation to the second derivative.

3.2. Fast Poisson Solver in 2D: Periodic Case

We consider Poisson's equation in the square $0<x,y<1$.

The 5-point finite difference approximation to $-u_{xx}-u_{yy}=f(x,y)$ is

$$(-u_{j+1k}+2u_{jk}-u_{j-1k})+(-u_{jk+1}+2u_{jk}-u_{jk-1})=h^2 f_{jk}. \tag{3.9}$$

The meshpoints lie on a regular grid. Each side of the square is devided into N+1 intervals of length $h=1/(N+1)$. There are N^2 meshpoints in the interior of the square. Here we begin with the *periodic* case, which corresponds to a discrete Fourier series. It is associated with the Fourier matrix F_{N+1}. Later the Dirichlet condition $u=0$ will be matched by a sine series and the Neumann condition $\frac{\partial u}{\partial n}$ by a cosine series.

The unknowns u_{jk} come in a natural order, starting with $u_{00}, u_{01}, \ldots, u_{0N+1}$

along the bottom of the square. That group will be denoted by $\underset{\sim}{U}_0$. Moving up the square we end with the unknowns along row N. By periodicity the group $\underset{\sim}{U}_{N+1}$ at the top of the square is the same as $\underset{\sim}{U}_0$ at the bottom. The right side $\underset{\sim}{f}$ will be ordered in the same way ($\underset{\sim}{f}_0, \underset{\sim}{f}_1, \ldots, \underset{\sim}{f}_N$) Then the 5-point schem becomes

$$\underset{\sim}{K}\,\hat{\underset{\sim}{U}} = \hat{\underset{\sim}{f}} \tag{3.10}$$

where

$$\underset{\sim}{K} = \begin{bmatrix} \underset{\sim}{T} & -\underset{\sim}{I} & 0 & . & -\underset{\sim}{I} \\ -\underset{\sim}{I} & \underset{\sim}{T} & -\underset{\sim}{I} & . & 0 \\ 0 & -\underset{\sim}{I} & \underset{\sim}{T} & . & 0 \\ . & . & . & . & -\underset{\sim}{I} \\ -\underset{\sim}{I} & 0 & 0 & -\underset{\sim}{I} & \underset{\sim}{T} \end{bmatrix}, \tag{3.11}$$

$$\hat{\underset{\sim}{U}} = \begin{bmatrix} \underset{\sim}{U}_0 \\ \underset{\sim}{U}_1 \\ . \\ \underset{\sim}{U}_N \end{bmatrix}, \qquad \text{and} \qquad \hat{\underset{\sim}{f}} = \begin{bmatrix} \underset{\sim}{f}_0 \\ \underset{\sim}{f}_1 \\ . \\ \underset{\sim}{f}_N \end{bmatrix}. \tag{3.12}$$

Here T is the same matrix as $\underset{\sim}{T}_{N+1}$ in equation (3.5) with d=4, and I is identity matrix. The order of $\underset{\sim}{T}$ and I is N+1, and $\underset{\sim}{K}$ is order of $(N+1)^2$. Equation (3.10) can also be written as

$$-\underset{\sim}{U}_{k+1} + \underset{\sim}{T}\,\underset{\sim}{U}_k - \underset{\sim}{U}_{k-1} = h^2 \underset{\sim}{f}_k, \quad k=0,1,\ldots,N. \tag{3.13}$$

This periodic matrix $\underset{\sim}{K}$ has one serious drawback. It is singular. Every row adds to zero. To make it nonsingular we can ground one or more nodes, which is the effect of a Dirichlet condition (u_0=0 will ground all boundary nodes).

3.2.1 The use of two-dimedsional Discrete Fourier Transform

Given complex function $h(k_1,k_2)$ defined over the two-dimensional grid $0\leq k_1\leq N_1-1$, $0\leq k_2\leq N_2-1$, we can define its two dimensional discrete Fourier transform as a complex function $H(n_1,n_2)$, defined over the same grid, by

$$H(n_1,n_2)\equiv\sum_{k_2=0}^{N_2-1}\sum_{k_1=0}^{N_1-1} h(k_1,k_2)e^{2\pi ik_2n_2/N_2}e^{2\pi ik_1n_1/N_1} \tag{3.14}$$

We can see that the two-dimensional FFT can be computed by taking one-dimensional FFTs sequentially on each index of the original function. Symbolically,

$$\begin{aligned} H(n_1,n_2) &= \text{FFT-on-index-1}\Big(\text{FFT-on-index-2}[h(k_1,k_2)]\Big) \\ &= \text{FFT-on-index-2}\Big(\text{FFT-on-index-1}[h(k_1,k_2)]\Big). \end{aligned} \tag{3.15}$$

Equation (3.15) provides a conveviet way for calculating the two-dimensional DFT but a more efficient way is a direct two-dimensional calculation. See appendix A.

Consider the solution of Poisson equation

$$\frac{\partial^2 u}{\partial^2 x^2}+\frac{\partial^2 u}{\partial^2 y}=\rho(x,y) \tag{3.16}$$

by the *finite difference method.* We represent the function $u(x,y)$ by its values at the discrete set of points

$$\begin{aligned} x_j &= x_0+j\Delta, \qquad j=0,1,\ \ldots\ ,J \\ y_l &= y_0+l\Delta, \qquad l=0,1,\ \ldots\ ,L \end{aligned} \tag{3.17}$$

where Δ is the *grid spacing.* From now on we will write $u_{j,l}$ for $u(x_j,y_l)$, and $\rho_{j,l}$ for $\rho(x_j,y_l)$. For (3.16) we substitute a finite-sifference representation

$$\frac{u_{j+1,l}-2u_{j,l}+u_{j-1,l}}{\Delta^2}+\frac{u_{j,l+1}-2u_{j,l}+u_{j,l-1}}{\Delta^2}=\rho_{j,l} \tag{3.18}$$

or equivalently

$$u_{j+1,l}+u_{j-1,l}+u_{j,l+1}+u_{j,l-1}-4u_{j,l}=\Delta^2\rho_{j,l} \tag{3.19}$$

The discrete inverse Fourier transform in both x and y is

$$u_{jl}=\frac{1}{JL}\sum_{m=0}^{J-1}\sum_{n=0}^{L-1}\hat{u}_{mn}e^{-2\pi ijm/J}e^{-2\pi iln/L} \tag{3.20}$$

Similarly,

$$\rho_{jl}=\frac{1}{JL}\sum_{m=0}^{J-1}\sum_{n=0}^{L-1}\hat{\rho}_{mn}e^{-2\pi ijm/J}e^{-2\pi iln/L} \tag{3.21}$$

If we substitute expressions (3.20) and (3.21) into (3.19), and we obtain

$$\frac{1}{JL}\sum_{m=0}^{J-1}\sum_{n=0}^{J-1}\hat{u}_{mn}[e^{-2\pi i(j+1)m/J}e^{-2\pi iln/L}+e^{-2\pi i(j-1)m/J}e^{-2\pi iln/L}$$

$$+e^{-2\pi ijm/J}e^{-2\pi i(l+1)n/L}+e^{-2\pi ijm/J}e^{-2\pi i(l-1)n/L}$$

$$-4e^{-2\pi ijm/J}e^{-2\pi iln/L}]=\Delta^2\frac{1}{JL}\sum_{m=0}^{J-1}\sum_{n=0}^{L-1}\hat{\rho}_{mn}e^{-2\pi ijm/J}e^{-2\pi iln/L} \tag{3.22}$$

or

$$\hat{u}_{mn}(e^{2\pi im/J}+e^{-2\pi im/J}+e^{2\pi in/L}+e^{-2\pi in/L}-4)=\hat{\rho}_{mn}\Delta^2 \tag{3.23}$$

or

$$\hat{u}_{mn}[2(\cos\tfrac{2\pi m}{J}+\cos\tfrac{2\pi n}{L}-2)]=\hat{\rho}_{mn}\Delta^2$$

then

$$\hat{u}_{mn}=\frac{\hat{\rho}_{mn}\Delta^2}{2(\cos\frac{2\pi m}{J}+\cos\frac{2\pi n}{L}-2)} \tag{3.24}$$

Thus the strategy for solving equation (3.19) by FFT techniques is:

i. Compute $\hat{\rho}_{mn}$ as the Fourier transform

$$\hat{\rho}_{mn} = \sum_{j=0}^{J-1} \sum_{l=0}^{L-1} \rho_{jl} e^{2\pi ijm/J} e^{2\pi iln/L} \tag{3.25}$$

ii. Compute $\hat{u}_{mn}$ from equation (3.24).

iii. Compute u_{ij} by the inverse Fourier transform (3.20).

The above procedure is valid for periodic boundary conditions. In other words, the solution satisfies

$$u_{jl} = u_{j+J,l} = u_{j,l+L}. \tag{3.26}$$

3.3. Fast Poisson Solver in 3D: Periodic Case

We consider the equation

$$U_{xx} + U_{yy} + U_{zz} = -F(x,y,z) \tag{3.27}$$

If the right side is $F = f(x,y)e^{iaz}$, the solutions are $U = u(x,y)e^{iaz}$. Substituting into (3.27) and cancelling e^{iaz} leaves

$$-\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} + a^2 u = f. \tag{3.28}$$

In the difference equation (3.25) this adds $h^2 a^2$ to the diagonals of $\underset{\sim}{T}$ and $\underset{\sim}{K}$.

Now comes the main idea. We know that $\underset{\sim}{T}$ is diagonalized by the Fourier matrix $\underset{\sim}{F}$. The matrix $\underset{\sim}{F}^{-1}\underset{\sim}{T}\underset{\sim}{F}$ is $\underset{\sim}{\Lambda}$, containing the $N+1$ eigenvalues of $\underset{\sim}{T}$. Our problem is to do something similar to the big matrix $\underset{\sim}{K}$, and the crucial first step is to transform each row separately:

$$\begin{bmatrix} F^1 & & & & \\ & F^1 & & & \\ & & F^1 & & \\ & & & . & \\ & & & & F^1 \end{bmatrix}\begin{bmatrix} T & -I & 0 & . & -I \\ -I & T & -I & . & 0 \\ 0 & -I & T & . & 0 \\ . & . & . & . & -I \\ -I & 0 & 0 & -I & T \end{bmatrix}\begin{bmatrix} F & & & & \\ & F & & & \\ & & F & & \\ & & & . & \\ & & & & F \end{bmatrix}$$

$$=\begin{bmatrix} \Lambda & -I & 0 & . & -I \\ -I & \Lambda & -I & . & 0 \\ 0 & -I & \Lambda & . & 0 \\ . & . & . & . & -I \\ -I & 0 & 0 & -I & \Lambda \end{bmatrix}. \tag{3.29}$$

This matrix is not fully diagonalized, but every block is now diagonal. Multiplying each row of the problem by $\underset{\sim}{F}^{-1}$ has changed

$$-\underset{\sim}{U}_{k+1}+\underset{\sim}{T}\underset{\sim}{U}_k-\underset{\sim}{U}_{k-1}=h^2\underset{\sim}{f} \quad \text{into} \quad -\underset{\sim}{V}_{k+1}+\underset{\sim}{\Lambda}\underset{\sim}{V}_k-\underset{\sim}{V}_{k-1}=h^2\underset{\sim}{c}_k. \tag{3.30}$$

The new unknown $\underset{\sim}{V}_k$ is the transform $\underset{\sim}{F}^{-1}\underset{\sim}{U}_k$ of the old unkown. The new right side $\underset{\sim}{c}_k$ is the transform $\underset{\sim}{F}^{-1}\underset{\sim}{f}_k$ of the old side. The middle term is $\underset{\sim}{F}^{-1}\underset{\sim}{T}\underset{\sim}{U}_k=\underset{\sim}{F}^{-1}\underset{\sim}{T}\underset{\sim}{F}\underset{\sim}{F}^{-1}\underset{\sim}{U}_k=\underset{\sim}{\Lambda}\underset{\sim}{V}_k$. Than the new equation in (3.30) is governed by the new matrix in (3.29):

$$\begin{bmatrix} \Lambda & -I & 0 & . & -I \\ -I & \Lambda & -I & . & 0 \\ 0 & -I & \Lambda & . & 0 \\ . & . & . & . & -I \\ -I & 0 & 0 & -I & \Lambda \end{bmatrix}\begin{bmatrix} V_0 \\ V_1 \\ V_2 \\ . \\ V_N \end{bmatrix}$$

$$=h^2\begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ . \\ c_N \end{bmatrix} \tag{3.31}$$

Only one more idea is needed. It is to look inside (3.31) for the jth componenents of all the V's and the jth components of all the c's. They satisfy

$$\begin{bmatrix} \lambda_j & -1 & 0 & . & -1 \\ -1 & \lambda_j & -1 & . & 0 \\ 0 & -1 & \lambda_j & . & 0 \\ . & . & . & . & -1 \\ -1 & 0 & 0 & -1 & \lambda_j \end{bmatrix} \begin{bmatrix} V_{0j} \\ V_{1j} \\ V_{2j} \\ . \\ V_{Nj} \end{bmatrix}$$

$$= h^2 \begin{bmatrix} c_{0j} \\ c_{1j} \\ c_{2j} \\ . \\ c_{Nj} \end{bmatrix} \tag{3.32}$$

This is a problem of size $N+1$ and not $(N+1)^2$. It is *one-dimensional difference* equation. There are $N+1$ of these easy problems, one for each j. Their solutions fill out the vector $\underset{\sim}{V}_k$, from which we recover the original unkowns $\underset{\sim}{U}_k = \underset{\sim}{F}\,\underset{\sim}{V}_k$. To express it differently, a reordering of the unkowns in the square array has renumbered the row and the columns of the big matrix in (3.32). Taking the first row of the every block, then all the second rows, and finally the last rows (and similarly for the columns) permutes the matrix into

$$\begin{bmatrix} T_0 & 0 & 0 & . & 0 \\ 0 & T_1 & 0 & . & 0 \\ 0 & 0 & T_2 & . & 0 \\ . & . & . & . & . \\ 0 & 0 & 0 & . & T_N \end{bmatrix} \quad \text{with } \underset{\sim}{T}_j \text{ as in (3.32).} \tag{3.33}$$

That matrix is still not diagonal but it is near enough. Each one-dimensional equation

(3.32) can be solved by the FFT or by elimination. With $d=4+a^2h^2$ on the main diagonal of $\underset{\sim}{K}$, we have reached a fast Poisson solver that needs only $N^2 \log N$ steps:

The nonsingular system $\underset{\sim}{K}\underset{\sim}{u}=\underset{\sim}{f}$ is solved in four stages:

1. Compute the vector $\underset{\sim}{c}_k=\underset{\sim}{F}^{-1}\underset{\sim}{f}_k$ for each block in f

2. Solve $N+1$ systems of the form (3.32) with $d_j=d-W^j-W^{-1}$

3. Reassemble those solutions in to the blocks $\underset{\sim}{V}_k$

4. Compute $\underset{\sim}{U}_k=\underset{\sim}{F}\underset{\sim}{V}_k$ to find each row of $\underset{\sim}{u}$.

Multiplication by $\underset{\sim}{F}^{-1}$ at step 1 and $\underset{\sim}{F}$ at step 4 requires $N\log_2 N$ multiplications by the Fast Fouier Transform, and they are done $N+1$ times—once for each block.

3.4. Fast Poisson Solver in 2D: Dirichlet Problem

After the periodic case it is easy to fix $u=0$ along the boundary. This is the Dirichlet problem and a nonzero boundary condition $u=u_0$ is not more diffcult. If u_{j0} or u_{0k} is prescribed in the finite difference equation, that term moves to the right side and affects only the vector $\underset{\sim}{f}$. The important changes are in $\underset{\sim}{K}$ and $\underset{\sim}{T}$:

$$\underset{\sim}{K}=\begin{bmatrix} T & -I & 0 & . & 0 \\ -I & T & -I & . & 0 \\ 0 & -I & T & . & 0 \\ . & . & . & . & -I \\ 0 & 0 & 0 & -I & T \end{bmatrix} \tag{3.34}$$

$$\text{with} \qquad \underset{\sim}{T}=\begin{bmatrix} 4 & -1 & 0 & . & 0 \\ -1 & 4 & -1 & . & 0 \\ 0 & -1 & 4 & . & 0 \\ . & . & . & . & -1 \\ 0 & 0 & 0 & -1 & 4 \end{bmatrix}. \tag{3.35}$$

These are now genuinely tridiagonal. K is block tridiogonal of order N. The -1's in the corners, which come from periodicity, have disappeared. The unkowns u_{jk} appear only at the N^2 interior meshpoints, and the boundary values are known. Thus a typical row of K for a point next to the boundary has only three -1's. Most rows still have four -1's and the corner meshpoints have only two—since their other neighbors are known from $u=u_0$.

We hope to keep the same algorithm for the new $\underset{\sim}{K}\underline{u}=\underline{f}$. However the eigenvectors of $\underset{\sim}{T}$ are no longer $(1, W^j, W^{2j}, \ldots)$. The nonperiodic matrix $\underset{\sim}{T}$ is not diagonalized by the Fourier matrix $\underset{\sim}{F}$. Fortunately the eigenvectors can still be found, but they are discrete sines instead of discrete exponentials. They are summarized by

$$\begin{bmatrix} 4 & -1 & 0 & . & 0 \\ -1 & 4 & -1 & . & 0 \\ 0 & -1 & 4 & . & 0 \\ . & . & . & . & -1 \\ 0 & 0 & 0 & -1 & 4 \end{bmatrix} \begin{bmatrix} \sin \pi jh \\ \sin 2\pi jh \\ \sin 3\pi jh \\ . \\ \sin N\pi jh \end{bmatrix}$$

$$= (4-2\cos \pi jh) \begin{bmatrix} \sin \pi jh \\ \sin 2\pi jh \\ \sin 3\pi jh \\ . \\ \sin N\pi jh \end{bmatrix}, \quad j=1,2,\ldots,N. \qquad (3.36)$$

The eigenvalues are $\lambda_j = 4-2\cos \pi jh$. Suppose the eigenvectors in (3.36) are the columns of an N *by* N "*sine matrix*" $\underset{\sim}{S}$. This matrix diagonalizes $\underset{\sim}{T}$, to give $\underset{\sim}{S}^{-1}\underset{\sim}{T}\underset{\sim}{S}=\underset{\sim}{\Lambda}$. The problem is again decoupled within each row, and the four steps of the periodic case are only slightly changed:

1. Compute the N-vector $\underset{\sim}{c}_k = \underset{\sim}{S}^{-1}\underset{\sim}{f}_k$ for each block in $\underset{\sim}{f}$

2. Solve N diagonal systems of the form (3.32), without -1's in the upper right and lower left corners

3. Reassemble those solutions into the N blocks $\underset{\sim}{V}_k$

4. Compute $\underset{\sim}{U}_k = \underset{\sim}{S}\,\underset{\sim}{V}_k$ to find each row of $\underset{\sim}{u}$.

The sine matrix $\underset{\sim}{S}$ takes the place of the Fourier matrix $\underset{\sim}{F}$, but otherwise the algorithm is unchanged.

3.4.1. The use of two-dimensional Discrete Sine Transforms

In the case of Dirichlet problem we can utilize a precedure similar to the one described in 3.2.1. but this time we need to dififne the two-dimensional discrete Sine transform $\hat{u}_{mn}$ of the solution u_{mn} over a discret set of point defined by (3.17). This transform is defined by

$$\hat{u}_{mn} = \sum_{j=1}^{J-1}\sum_{l=1}^{L-1} u_{jl}\sin\frac{\pi jm}{J}\sin\frac{\pi ln}{L} \tag{3.37}$$

Consider now a Dirichlet boundary conditon $u=0$ on the rectangular boundary. Instead of expansions (3.20), (3.21) we now need an expansion in sine waves:

$$\rho_{jl} = \frac{2}{J}\frac{2}{L}\sum_{m=1}^{J-1}\sum_{n=1}^{L-1}\hat{\rho}_{mn}\sin\frac{\pi jm}{J}\sin\frac{\pi ln}{L} \tag{3.38}$$

$$u_{jl} = \frac{2}{J}\frac{2}{L}\sum_{m=1}^{J-1}\sum_{n=1}^{L-1}\hat{u}_{mn}\sin\frac{\pi jm}{J}\sin\frac{\pi ln}{L} \tag{3.39}$$

This satisfies the boundary conditions that $u=0$ at $j=0,J$ and at $l=0,L$. If we

substitute (3.38) and (3.39) into equation (3.19), we obtain

$$\hat{u}_{mn}[\sin\frac{\pi ln}{L}(\sin\frac{\pi(j+1)m}{J}+\sin\frac{\pi(j-1)m}{J})+\sin\frac{\pi jm}{J}(\sin\frac{(l+1)n}{L}+\sin\frac{\pi(l-1)n}{L})$$

$$-4\sin\frac{\pi jm}{J}\sin\frac{\pi ln}{L}]=\Delta^2\hat{\rho}_{mn}\sin\frac{\pi jm}{J}\sin\frac{\pi ln}{L} \tag{3.40}$$

since,

$$\sin(A+B)=\sin A\cos B+\cos A\sin B$$

the equation (3.40) can be written in the form of

$$\hat{u}_{mn}[2\cos\frac{\pi m}{J}+2\cos\frac{\pi n}{L}-4]=\Delta^2\hat{\rho}_{mn}$$

or

$$\hat{u}_{mn}=\frac{\hat{\rho}_{mn}\Delta^2}{2(\cos\frac{\pi m}{J}+\cos\frac{\pi n}{L}-2)} \tag{3.41}$$

we find that the solution procedure parallels that for periodic boundary conditions:

i. Compute $\hat{\rho}_{mn}=\sum_{j=1}^{J-1}\sum_{l=1}^{L-1}\rho_{jl}\sin\frac{\pi jm}{J}\sin\frac{\pi ln}{L}$ (3.42)

ii. Compute $\hat{u}_{mn}$ from equation (3.41)

iii. Compute u_{jl} by the inverse sine transform (3.39).

A subroutine in FORTRAN77 which calculates the 2D Discrete Sine transform of a JxL array is discribed in appendix B.

4. A boundary value problem

In this section we illustrate the use of FFT (or the Fast Sine Transform to be exact) by applying them to the Dirichlet problem.

$$\frac{\partial^2 u(x,y)}{\partial x^2}+\frac{\partial^2 u(x,y)}{\partial x^2}=0 \qquad (0<x<1,\ 0<y<1) \tag{4.1}$$

$$u(x,0)=0, \qquad u(x,1)=0 \qquad (0<x<1) \tag{4.2}$$

$$u(0,y)=0, \qquad u(1,y)=1 \qquad (0<y<1) \tag{4.3}$$

Separation of variables, with $u=X(x)\,Y(y)$, leads to the Sturm-Liouville problem

$$Y''(y)+\lambda\,Y(y)=0, \qquad Y(0)=0, \qquad Y(1)=0, \tag{4.4}$$

whose eigenvalues and eigenfunctions are

$$\lambda_m=m\pi, \quad Y(y)=\sin\ m\pi y, \quad m=1,2,\ \ldots\ . \tag{4.5}$$

In addition we have

$$X''(x)-\lambda^2 X(x)=0, \qquad X(0)=0 \tag{4.6}$$

or

$$X(x)=A_m \sinh\ m\pi x, \qquad m=1,2,\ \ldots \tag{4.7}$$

Thus a solution satisfying all these conditions is

$$u(x,y)=A_m \sinh\ m\pi x\ \sin\ m\pi y, \qquad m=1,2,\ \ldots \tag{4.8}$$

To satisfy the last condition, $u(1,y)=1$, we must first use the principle of the superposition to obtain the solution

$$u(x,y)=\sum_{m=1}^{\infty} A_m \sinh\ m\pi x\ \sin\ m\pi y, \tag{4.9}$$

and from $u(1,y)=1$ we obtain

$$1=\sum_{m=1}^{\infty} A_m \sinh\ m\pi\ \sin\ m\pi y\ . \tag{4.10}$$

Using the theory of Fourier series,

$$A_m \sinh\, m\pi = 2\int_0^1 \sin\, m\pi y\, dy = \frac{2(1-\cos\, m\pi)}{m\pi}\,, \tag{4.11}$$

and

$$A_m = \frac{2(1-\cos\, m\pi)}{m\pi \sinh\, m\pi}\,. \tag{4.12}$$

From (4.10) and (4.12), we obtain the general solution

$$u(x,y) = \sum_{m=1}^{\infty} \frac{2(1-\cos\, m\pi)}{m\pi \sinh\, m\pi} \sinh\, m\pi x \sin\, m\pi y \tag{4.13}$$

In oder to increase the accuracy of the numerical calculation of (4.13) at large values of m we note that

$$\frac{\sinh\, m\pi x}{\sinh\, m\pi} = \frac{e^{m\pi(x-1)}(1-e^{-2m\pi x})}{1-e^{-2m\pi}}\,. \tag{4.14}$$

4.1. *Simultaneous Over-Relaxation* (*SOR*)

In this section we discuse a standard numerical method for obtaining the solution of the problem (4.1)-(4.3).

Consider the dicrete set of points (3.17). Applying finite differecing to the diferential eqation (4.1) we obtain

$$u_{j+1,l} + u_{j-1,l} + u_{j,l+1} + u_{j,l-1} - 4u_{j,l} = 0\,. \tag{4.15}$$

For each one of the interior points, *i.e.*

$$j = 1,2,\ldots,J-1$$
$$l = 1,2,..,L-1 \tag{4.16}$$

This requires the solution of a system of the form

$$\underline{\underline{A}}\,\underline{u}=\underline{b}\ , \tag{4.17}$$

where (4.17) is a square matrix of size (L−1)x(J−1), and the one dimensional array $\underline{u}$ contains the (L−1)x(J−1) unknowns discrete values of the solution. By proper ordering of the entries of $\underline{u}$ the matrix $\underline{\underline{A}}$ can be made into a band matrix of width $2J-1$, (see figure 4.1.) assuming that $J \leq L$.

For a dense grid (large L,J) direct solution of (4.17) is not practical. In this case an iteration procedure of the Gauss-Seidel type is desired. A very efficient method is the one known as SOR which can be summarized as follows.

We start with a guess $u_{j,l}{}^{old}$ and we update the values by

$$u_{j,l}{}^{new}=w\ u_{j,l}{}^{*}+(1-w)\ u_{j,l}{}^{old} \tag{4.18}$$

where

$$u^{*}_{j,l}=\frac{1}{4}[(u_{j+1,l}+u_{j-1,l}+u_{j,l+1}+u_{j,l-1})]_{old} \tag{4.19}$$

and w is the overrelaxation factor. Note that for $w=1$, equation (4.18) in the Gauss-Seidel iteration .

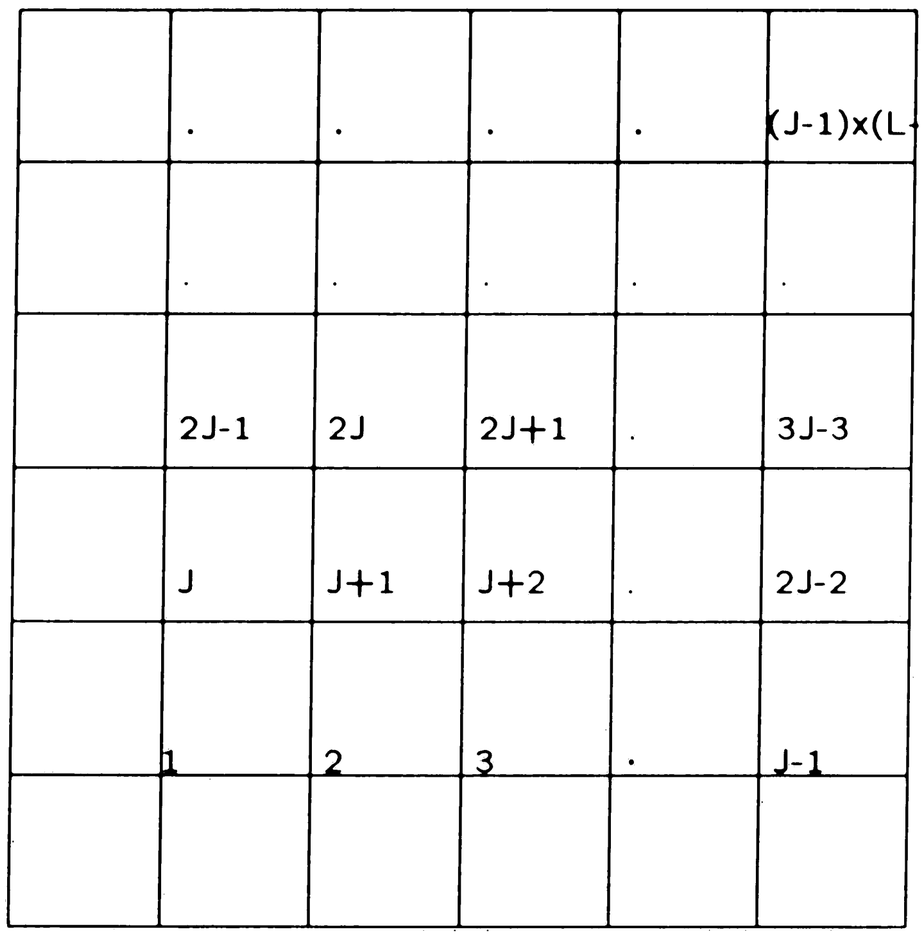
If $(\rho_{Jacobi})^2$ is the spectral radius of the Gauss-Seidel iteration then an *optimal* choice for w is given by

$$w=\frac{2}{1+\sqrt{1-(\rho_{Jacobi})^2}}\ . \tag{4.20}$$

Some SOR routines use Chebyshev acceleration in the refining the initial guess of ρ_{Jacobi} (or w). The subroutine SOR, explained in appendix C, can be used for equations more general than (4.15). Any equation with variable coefficient which has a finite difference approximation of the form

$$a_{j,l}u_{j+1,l}+b_{j,l}u_{j-1,l}+c_{j,l}u_{j,l+1}+d_{j,l}u_{j,l-1}+e_{j,l}u_{j,l}=f_{j,l}. \tag{4.21}$$

can be solved by SOR. The storage requirmets are 7 matrices, each of which has $J-1$ columns and $L-1$ rows.



$$\underset{\sim}{u}=\begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ \cdot \\ u_{(J-1)\text{x}(L-1)} \end{bmatrix}$$

Figure 4.1. Finite difference net for a large Dirichlet problem.

4.2. Use of Discrete Sine Transform

In this section we will modify the method described in 3.4.1. to solve the boudary value problem described by (4.1.)-(4.3.). We now have a homegeneous equation but inhomegeneous boundary data at $x=1$.

The finite difference approximation of (4.1.) at all points other than at $j=J-1$ is the same as the one for the case of a completely homegeneous problem. At $j=J-1$ we have

$$u_{J-2,l}+u_{J-1,l+1}+u_{J-1,l-1}-4u_{J-1,l}=-1 \tag{4.22}$$

for $l=1,2,\ldots,L-1$.

This problem is identical to having over the described grid

$$\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}=0 \tag{4.23}$$

with homegeneous boundary data

$$u(0,y)=u(1,y)=u(x,y)=u(x,1)=0 \tag{4.24}$$

and

$$\rho(x,y)=\begin{cases}-1, & \text{for } j=J-1 \text{ and } l=1,2,\ldots,L-1\\ 0, & \text{otherwise}\end{cases} \tag{4.25}$$

We can now compute the SFT $\hat{\rho}$ of ρ given by (4.23) using the SIN1. Expanding ρ, u in terms of their sine transform as explaned in (3.38)-(3.39) and substituting them in (4.23) we obtain

$$\hat{u}_{mn}=\frac{\hat{\rho}_{mn}\Delta^2}{2(\cos\frac{2\pi m}{J}+\cos\frac{2\pi n}{L}-2)}. \tag{4.26}$$

Taking the inverse of $\hat{u}_{mn}$ produces the solution of the problem.

35. Comparison of calculated results

The analytical solution of the boundary value problem (4.1.)-(4.3.) is given in table 1. These results are given with 6 significent figure accuracy and have been obtained on a Zenith 158 PC with a 8087 numeric data processor using WATFOR77 fortran.

$y\backslash x$	0	1/8	1/4	3/8	1/2	5/8	3/4	7/8	1
0	0.	0.000000	0.000000	0.000000	0.000001	0.000001	0.000002	0.000004	
1/8	0.	0.017091	0.036983	0.063124	0.100708	0.159435	0.262589	0.482912	1.
1/4	0.	0.031478	0.067972	0.115431	0.182029	0.280451	0.432030	0.668954	1.
3/8	0.	0.040997	0.088343	0.149293	0.232910	0.350708	0.515778	0.736447	1.
1/2	0.	0.044316	0.095414	0.160925	0.250000	0.373257	0.540530	0.754270	1.
5/8	0.	0.040997	0.088343	0.149293	0.232910	0.350708	0.515778	0.736447	1.
3/4	0.	0.031478	0.067972	0.115430	0.182029	0.280450	0.432029	0.668953	1.
7/8	0.	0.017091	0.036983	0.063123	0.100708	0.159434	0.262587	0.482910	1.
1	0.	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	

Table 1. Exact solution.

The results obtained using SOR over a 8x8 grid (SOR8) are given in table 2, and the results for SOR over a 16x16 grid (SOR16) are given in table 3. It is clear that as the grid becomes denser the accuracy of SOR increaces. In SOR8 the maximum relative error* is about 0.026 and it occurs at the point (x=5/8 ; y=1/8 , 7/8).

In SOR16 the maximum relative error is about 0.008 and it occurs at the point (x=6/8 ; y=1/8 , 7/8).

$y\backslash x$	0	1/8	1/4	3/8	1/2	5/8	3/4	7/8	1
0	0.	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
1/8	0.	0.017413	0.037683	0.064373	0.102941	0.163567	0.269302	0.482587	1.
1/4	0.	0.031968	0.068947	0.116869	0.183823	0.282028	0.431052	0.661045	1.
3/8	0.	0.041515	0.089266	0.150332	0.233456	0.349667	0.511836	0.730543	1.
1/2	0.	0.044825	0.096273	0.161734	0.249999	0.371354	0.536078	0.749291	1.
5/8	0.	0.041515	0.089266	0.150332	0.233456	0.349667	0.511836	0.730543	1.
3/4	0.	0.031968	0.068947	0.116869	0.183823	0.282028	0.431052	0.661045	1.
7/8	0.	0.017413	0.037683	0.064373	0.102941	0.163567	0.269302	0.482587	1.
1	0.	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	

Table 2. SOR8.

$y\backslash x$	0	1/8	1/4	3/8	1/2	5/8	3/4	7/8	1
0	0.	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
1/8	0.	0.017171	0.037155	0.063433	0.101276	0.160572	0.264768	0.482830	1.
1/4	0.	0.031605	0.068221	0.115804	0.182516	0.280925	0.431779	0.666475	1.
3/8	0.	0.041135	0.088585	0.149569	0.233061	0.350431	0.514685	0.734864	1.
1/2	0.	0.044453	0.095642	0.161142	0.249999	0.372734	0.539325	0.752998	1.
5/8	0.	0.041135	0.088585	0.149569	0.233061	0.350431	0.514685	0.734864	1.
3/4	0.	0.031605	0.068221	0.115804	0.182516	0.280925	0.431779	0.666475	1.
7/8	0.	0.017171	0.037155	0.063433	0.101276	0.160572	0.264768	0.482830	1.
1	0.	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	

Table 3. SOR16.

* *relative error*$=|(u_{SOR}-u_{Exact})/u_{Exact}|$.

The results obtained using SFT over a 8x8 grid (SFT8) are given in table 4, and the results for SFT over a 16x16 grid (SFT16) are given in table 5. The maximum relative errors in these cases (and indeed almost all relative errors) are identical to the ones for SOR.

$y\backslash x$	0	1/8	1/4	3/8	1/2	5/8	3/4	7/8	1
0	0.	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
1/8	0.	0.017413	0.037683	0.064373	0.102941	0.163568	0.269302	0.482588	1.
1/4	0.	0.031969	0.068947	0.116869	0.183824	0.282029	0.431054	0.661047	1.
3/8	0.	0.041515	0.089267	0.150332	0.233456	0.349669	0.511837	0.730544	1.
1/2	0.	0.044826	0.096274	0.161734	0.250000	0.371355	0.536080	0.749293	1.
5/8	0.	0.041515	0.089267	0.150332	0.233456	0.349669	0.511837	0.730544	1.
3/4	0.	0.031969	0.068947	0.116869	0.183824	0.282029	0.431054	0.661047	1.
7/8	0.	0.017413	0.037683	0.064373	0.102941	0.163568	0.269302	0.482588	1.
1	0.	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	

Table 4. SFT8.

$y\backslash x$	0	1/8	1/4	3/8	1/2	5/8	3/4	7/8	1
0	0.	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
1/8	0.	0.017170	0.037155	0.063433	0.101276	0.160572	0.264768	0.482831	1.
1/4	0.	0.031603	0.068221	0.115804	0.182516	0.280925	0.431780	0.666475	1.
3/8	0.	0.041132	0.088586	0.149570	0.233062	0.350431	0.514686	0.734865	1.
1/2	0.	0.044450	0.095643	0.161143	0.250000	0.372735	0.539326	0.752999	1.
5/8	0.	0.041132	0.088586	0.149570	0.233062	0.350431	0.514686	0.734865	1.
3/4	0.	0.031603	0.068221	0.115804	0.182516	0.280925	0.431780	0.666475	1.
7/8	0.	0.017170	0.037155	0.063433	0.101276	0.160572	0.264768	0.482831	1.
1	0.	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	

Table 5. SFT16.

The maximum relative errors for the 4 tables described above are tabulated for convenience in table 6 together with the execution times required to calculate the complete tables.

Method	Max. relative error	Execution time
SOR8	0.026	00:09.18
SOR16	0.008	01:03.33
SFT8	0.026	00:04.28
SFT16	0.008	00:12.47

Table 6.

The maximum relative errors for the 4 tables described above are tabulated for convenience in table 6 together with the execution times required to calculate the complete tables.

Method	Max. relative error	Execution time
SOR8	0.026	00:09.18
SOR16	0.008	01:03.33
SFT8	0.026	00:04.28
SFT16	0.008	00:12.47

Table 6.

Appendix A

Subroutines for Fast Fourier Transform (FFT)

For one dimensional FFT of an array DATA with NN entries we used SUBROUTINE FOUR1(DATA, NN, ISIGN) which is given in [1]. This subroutine together with all other subroutines and programs used in this thesis are kept in a file in the Department of Mechanical Engineering and Mechanics under the author's name. In the subroutine FOUR1 the value of ISIGN determines wether the transform is forward or backward. For ISIGN$=+1$ DATA is repraced by its dicrete Fourier transform. For ISIGN$=-1$, DATA is replaced by NN times its inverse discrete Fourier transform. NN must be a power of 2.

For a 2D FFT one could write a program based on equation (3.15) (in fact this is what we do in the case of Fast Sine Transform) but it is more efficient to use a direct two-dimensional calculation. Such a direct routine is SUBROUTINE FOURN(DATA, NN, NDIM, ISIGN). Which is given in [1]. This subroutine replaces the NDIM-dimension array DATA by its discrete Fourier transform, if ISIGN is inputed as $+1$. NN is an integer array of length NDIM and contains the length of each dimension. All lengths must be powers of 2. If ISIGN$=-1$, DATA is replaced by its inverse trnsform times the product of the lengths of all dimensions.

Appendix B

Sobroutine for Fast Sine Transform (FST)

For a one dimensional FST of a set of N real-valued data points stored in array Y, we used SUBROUTINE SINFT(Y, N) which is given in [1]. In the subroutine the number N must be a power of 2. On exit Y is repraced by its transform. The program without changes, also calculates the inverse sine transform, but in this case the output array should be multiplied by 2/N.

For a 2D FST we used SUBROUTINE SIN1(F, LL, JJ, FHH) for forward transform and SUBROUTINE SIN2(FHH, LL, JJ, F) for backword transform. These routines are written by the author and are availeble in the file kept in the Department of Machanical Engineering and Machanics as explained in appendix A. In the subroutine SIN1, F is a matrix with LL rows and JJ columns. The subroutine first tranforms the F matrix column by column by calling the SUBROUTINE SINFT(Y, N) which is explained in the previous pragraph. Then it transforms the F matrix row by row again by using the SUBROUTINE SINFT(Y, N). The transform is stored in FHH. Similarly SUBROUTINE SIN2(FHH, LL, JJ, F) calculates the inverse transform of the matrix FHH, which has LL rows and JJ columns, and stores it in F matrix. Here again the transform is being done first column by column then row by row by using the SUBROUTINE SINFT(Y, N) which is explained above.

Appendix C

Subroutine for Simultaneus Over-Relaxation (SOR)

The solution of the finite difference approximation equation (4.21) is obtained using SUBROUTINE SOR(A, B, C, D, E, F, U, JMAX, RJAC). Matrices A, B, C, D, E, F, U are of size JMAX x JMAX and contain the coefficients of the equation (4.21) over the square grid JMAX x JMAX. On output the solution is stored in matrix U of the size JAX x JAX. On input U contains the initial guess of the solution. RJAC is input as the spectral radius of the Jacobi iteration, or an estimate of it. This subroutine uses Chebyshev accelaration in the simultaneus over-relaxation.

Bibliography

[1] PRESS, William H., FLANNERY, Brian P., TEUKOLSKY, Saul A., and VETTERLING, William T. Numerical Recipes, The Art of Scientific Computing. Cambridge University Press, Cambridge, 1986.

[2] STRANG, Gilbert. Introduction to Applied Mathematics. Wellesley-Cambrige Press., Wellesley, MA, 1986.

[3] BRIGHAM, E. Oran. The Fast Fourier Transform. Englewood Cliffs, N.J.:Prentice Hall, 1974.

[4] KRISHNAMURTHY, E. V. Error-Free Polynomial Matrix Computations. Springer-Verlag New york Inc. New York, 1985.

[5] ELLIOTT, D. F., and RAO, K. R. Fast Transforms:Algorithms, Analyses, Applications New York:Academic Press,1982.

[6] PICKERING, Morgan. An Introduction to Fast Fourier Transform Methods for Partial Differential Equations, with Applications. John Willey & Sons Inc., New York, 1986.

[7] J. W. Cooley, P. A. W. Lewis and P. D. Welch. J. Sound Vib. (1970) 12(3) 315-337. The Fast Fourier Transform Algorithm: Programming considerations in the Calculation of Sine, Cosine and Laplace Transforms.

Resume

Mustafa Kizilkaya

Center for the Application of Mathematics

Lehigh University

Bethlehem, PA 18015

or

M. Kazim Baser Mahallesi

535/2 No 1-A

ADANA-TURKEY

Date and Place of Birth: March 20, 1961, Adana, TURKEY.

Education: M.S., expected June, 1989, Lehigh University, Bethlehem, Pa.

Major: Applied Mathematics.

Thesis: Some Applications of Fast Fourier Transform .

B.S. June, 1983, Firat University, Elazig, Turkey.

Major: Mathematics.

Experience: 1983-86. Teaching Assistant, Department of Mathematics,

Inonu University, Malatya,TURKEY.

Received full scolarship from the Turkish Government

for M.S. and Ph.D. studies.